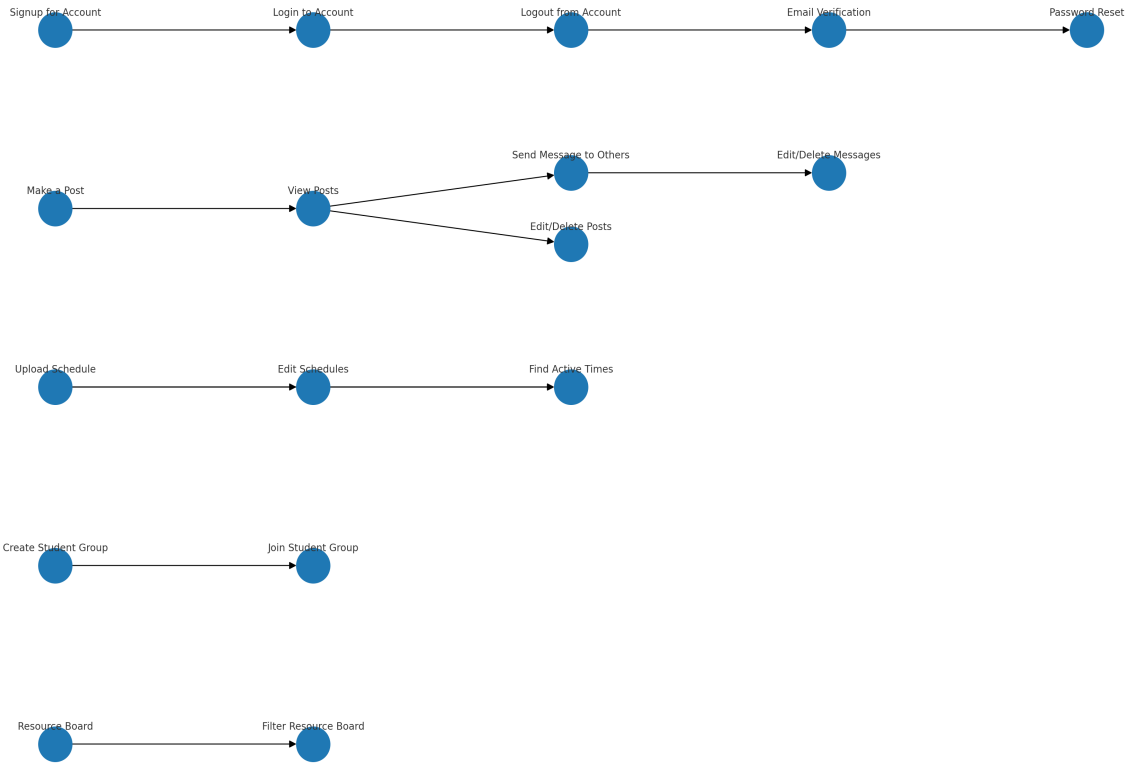

Task Dependencies and Network Diagram The tasks in our project follow a number of natural dependencies. For example, a user cannot log in without first signing up, and they cannot view posts unless posts already exist. These dependencies form the basis of our network diagram, where each task flows into the ones that rely on it. This helps determine the proper sequence of work and prevents feature blockages. Critical Path The critical path represents the longest chain of dependent tasks that determines the overall completion timeline. Based on our dependency flows, likely critical paths include: - Signup → Login → Logout → Email Verification → Password Reset - Make a Post → View Posts → Send Message → Edit/Delete Messages - Upload Schedule → Edit Schedules → Find Active Times Any delay in these chains would directly push the project schedule back. Keeping the Sprint on Schedule To stay on schedule, we ensure that each team member selects tasks that do not rely on unfinished dependencies. Tasks must be clearly defined, and subtasks broken down to avoid ambiguity. Blocking relationships are avoided when assigning work. For example, it would not make sense to work on View Posts before Make a Post is completed. What Went Wrong and Lessons Learned We were unable to finish the Upload Schedule task during the sprint. The complexity of the task was underestimated and required more backend logic than initially anticipated, as documented in PB.md. The main lesson learned is the importance of breaking down large tasks into smaller, clear subtasks and identifying complexity early during sprint planning.